### BUILDING AND DEPLOYING MICROSERVICES STACK WITH DOCKER COMPOSEIntroduction & Learning Objectives

# Chapter Overview

### BUILDING AND DEPLOYING MICROSERVICES STACK WITH DOCKER COMPOSEKnowledge Check

Congratulations on completing Chapter 8 - Building and Deploying Microservices Stack with Docker Compose. Take this quiz to check your understanding of the concepts you've learned about so far.

BUILDING AND DEPLOYING MICROSERVICES STACK WITH DOCKER COMPOSEIntroduction & Learning Objectives Chapter Overview BUILDING AND DEPLOYING MICROSERVICES STACK WITH DOCKER COMPOSEKnowledge Check Congratulations on completing Chapter 8 - Building and Deploying Microservices Stack with Docker Compose. Take this quiz to check your understanding of the concepts you've learned about so far. Question 8.1 You can apply docker-compose spec in Kubernetes as is. True or False? A. True B. False

ChatGPT

B. False

The statement is False.

While both Docker Compose and Kubernetes are tools used for container orchestration, they have different formats and purposes. Docker Compose is designed to define and manage multi-container applications, typically for development or testing environments, on a single host. It uses a YAML file (docker-compose.yml) to describe the services, networks, and volumes required for the application.

On the other hand, Kubernetes is a more powerful container orchestration system used for managing containerized applications in production environments. Kubernetes uses its own YAML manifest files (usually with .yaml or .yml extensions) to define the desired state of the application and its components, including pods, services, deployments, and more.

While there are similarities between the services and components defined in Docker Compose YAML files and Kubernetes YAML files, the formats and options are not directly interchangeable. Therefore, you cannot apply a docker-compose spec directly in Kubernetes as it is. The Kubernetes YAML files need to be specifically written or converted to the appropriate Kubernetes resources.

However, some tools and utilities exist to help with the migration or conversion of Docker Compose files to Kubernetes manifests, making it easier to transition from local development using Docker Compose to running applications in a Kubernetes cluster. But it's important to understand that the YAML files used in Docker Compose and Kubernetes have different structures and are meant for different purposes.



Question 8.2 If you use the --link option, it will enable DNS-based service discovery. True or False? A. True B. False

ChatGPT

B. False

The statement is False.

The **--link** option in Docker Compose is used to link containers together within the same Docker Compose project. It establishes a secure tunnel between the linked containers and sets environment variables in the linked container to allow easy communication between them. However, it does not enable DNS-based service discovery.

In previous versions of Docker, using **--link** allowed containers to communicate with each other using short names based on the linked container names, and Docker provided environment variables with connection information. However, the **--link** option has been deprecated in favor of using user-defined networks in Docker Compose.

To enable DNS-based service discovery within Docker Compose, you should use user-defined networks. When containers are connected to the same user-defined network, they can discover and communicate with each other using their service names or container names as DNS names. This feature is more flexible and recommended over the deprecated **--link** option.

To create and use user-defined networks in Docker Compose, you define the networks section in your **docker-compose.yml** file and add services to that network. This allows the services to communicate with each other using their service names as DNS names, making service discovery easier and more manageable.

In summary, the **--link** option does not enable DNS-based service discovery. For service discovery within Docker Compose, it is recommended to use user-defined networks.



Question 8.3 In which of the following languages is Docker Compose written in? A. COMPML B. YAML C. Golang D. None of the above

ChatGPT

C. Golang

Docker Compose is primarily written in the Go programming language (Golang). Go is an open-source programming language developed by Google, and it is known for its simplicity, efficiency, and fast compilation. Docker Compose leverages the power of Go to handle the orchestration and management of multi-container applications described in the **docker-compose.yml** file.

YAML (option B) is the file format used for defining the services, networks, and volumes in the **docker-compose.yml** file. YAML is a human-readable data serialization format that is often used for configuration files.

COMPML (option A) is not a known programming language or file format, and it is not related to Docker Compose.

In summary, Docker Compose is primarily written in the Go programming language (Golang), and the **docker-compose.yml** file uses YAML as the format for defining the composition of multi-container applications.



Question 8.4 You can replace docker-compose with Dockerfile. True or False? A. True B. False

ChatGPT

B. False

The statement is False.

Docker Compose and Dockerfile serve different purposes in the container ecosystem and cannot be directly replaced with each other.

1. **Docker Compose:** Docker Compose is a tool used for defining and managing multi-container applications. It allows you to define the services, networks, and volumes required for an application in a single YAML file (**docker-compose.yml**). Docker Compose simplifies the process of running and managing multi-container applications by providing a simple and consistent way to define the entire application stack.
2. **Dockerfile:** A Dockerfile is a script used to build a Docker image. It contains instructions on how to create a container image from a base image, install dependencies, copy files into the image, and configure the container. The Dockerfile is used with the **docker build** command to create a custom image that can be run as a container.

While both Docker Compose and Dockerfile are used in the context of containerization, they have different roles:

* Docker Compose focuses on the orchestration and management of multi-container applications, defining the composition of the application stack, and handling inter-container communication.
* Dockerfile focuses on building custom container images, specifying the contents and configuration of the image that will be used to run the container.

In some cases, you may use both Docker Compose and Dockerfile together to manage the entire container lifecycle. Docker Compose defines the application stack, and within each service's definition, you can specify the Dockerfile that should be used to build the image for that service.

In summary, Docker Compose and Dockerfile are complementary tools, each serving its specific purpose, and they are not interchangeable. Docker Compose is for defining multi-container applications and managing their interactions, while Dockerfile is for building custom container images.



Question 8.5 Which of the following features makes docker-compose a safe tool to deploy to a dev environment by setting up a process to deploy at regular intervals? A. Autosync B. Idempotence C. Service discovery D. Scalability

ChatGPT

B. Idempotence

The feature that makes docker-compose a safe tool to deploy to a dev environment by setting up a process to deploy at regular intervals is "Idempotence."

Idempotence refers to the property of an operation that, when applied multiple times, produces the same result as if it were applied only once. In the context of deployment, an idempotent process ensures that applying the deployment configuration multiple times does not lead to unintended side effects or inconsistencies. It allows for predictable and safe deployment updates, even if the process is executed repeatedly.

Docker Compose is designed to be idempotent, meaning that running the same **docker-compose up** command multiple times will not result in additional or conflicting containers being created. If the containers specified in the **docker-compose.yml** file are already running, Docker Compose will simply ensure that they are in the desired state, making it safe to run the deployment process at regular intervals.

By leveraging the idempotent nature of Docker Compose, developers can automate deployment in a development environment without worrying about unintended consequences or issues caused by repeated deployment actions.

Options A, C, and D are not directly related to the feature that makes docker-compose a safe tool for deploying in a dev environment with regular intervals. Here's a brief explanation of those options:

A. Autosync: This term is not directly related to a specific feature of Docker Compose. It could refer to a mechanism that automatically synchronizes changes between multiple instances or environments.

C. Service discovery: Service discovery is the process of automatically finding and registering the network location of services in a distributed system. While Docker Compose allows containers to communicate with each other using service names as DNS names, it is not the primary feature that ensures safe deployment in a dev environment.

D. Scalability: Scalability refers to the ability of a system to handle growing amounts of work or its potential to be enlarged to accommodate that growth. While Docker Compose can be used to deploy multi-container applications, the concept of scalability is more related to how those applications are designed to handle increasing workloads and resource demands.



Question 8.6 Which of the following commands should be used to bring up a docker-compose stack? A. docker-compose up -d B. docker run -f docker-compose.yml C. docker-compose run -fd docker-compse.yml D. None of the above

ChatGPT

A. docker-compose up -d

The correct command to bring up a Docker Compose stack is:

Copy code

docker-compose up -d

This command reads the **docker-compose.yml** file (the default filename) in the current directory and creates and starts the services defined in the file. The **-d** option stands for "detached" mode, which means the containers will run in the background, allowing you to continue using the terminal without being attached to the container's logs.

Option B (**docker run -f docker-compose.yml**) is not the correct command to bring up a Docker Compose stack. The **docker run** command is used to run a single container, and the **-f** option specifies a different Docker Compose file to be used, but it does not trigger the full stack deployment defined in the **docker-compose.yml** file.

Option C (**docker-compose run -fd docker-compse.yml**) contains a typo in the filename and is also not the correct command. The **-f** option should be used with **docker-compose up**, not with **docker-compose run**.

Option D (**None of the above**) is incorrect because option A (**docker-compose up -d**) is the correct command to bring up a Docker Compose stack, as described earlier.



Question 8.7 How do you control the start up order of containers? A. Using the depends\_on option B. Using the links option C. Using the wait\_for option D. None of the above

ChatGPT

A. Using the depends\_on option

You control the start-up order of containers in Docker Compose by using the **depends\_on** option. The **depends\_on** option allows you to specify the order in which services should be started or the dependencies between services.

Here's an example of how to use **depends\_on** in a **docker-compose.yml** file:

yamlCopy code

version: '3' services: webapp: build: ./webapp depends\_on: - database database: image: mysql:latest

In this example, the **webapp** service depends on the **database** service. When you use **docker-compose up**, Compose will first start the **database** service, and then it will start the **webapp** service once the **database** service is up and running.

It's important to note that **depends\_on** only controls the start-up order of containers. It does not wait for the containers to be fully "ready" before proceeding to start the dependent services. If your application requires a service to be fully initialized before another service starts, you may need to implement additional mechanisms for synchronization, such as using a health check or a custom script to wait for the readiness of the dependent service.

Options B and C are not correct:

* Option B (**Using the links option**) is incorrect because the **links** option is a legacy feature that allows containers to communicate with each other by updating the **/etc/hosts** file. It is not used for controlling the start-up order.
* Option C (**Using the wait\_for option**) is not a standard option in Docker Compose for controlling the start-up order of containers.

Option D (**None of the above**) is incorrect because option A (**Using the depends\_on option**) is the correct answer for controlling the start-up order of containers in Docker Compose.

### QUIZ RESULTS

YOU GOT 6 OUT OF 7 CORRECT: 86%

7 7 6 1

* TOTAL (7)
* ANSWERED
* CORRECT
* INCORRECT

Retake Quiz Review All Question Results

[Continue to Next Section](https://trainingportal.linuxfoundation.org/learn/course/containers-for-developers-and-quality-assurance/building-and-deploying-microservices-stack-with-docker-compose/knowledge-check/completed)

# Question 8.3

In which of the following languages is Docker Compose written in?

Incorrect Answer

* **A.** COMPML
* **B.** YAML

Correct Answer

* **C.** Golang

Your Answer:

Incorrect

* **D.** None of the above

Next Question

1

2

3

4

5

6

7